





HD28

.M414

no. 1980-88

MAR 16 1988

LIBRARIES

WORKING PAPER
ALFRED P. SLOAN SCHOOL OF MANAGEMENT

MODELING THE DYNAMICS OF
SOFTWARE PROJECT MANAGEMENT

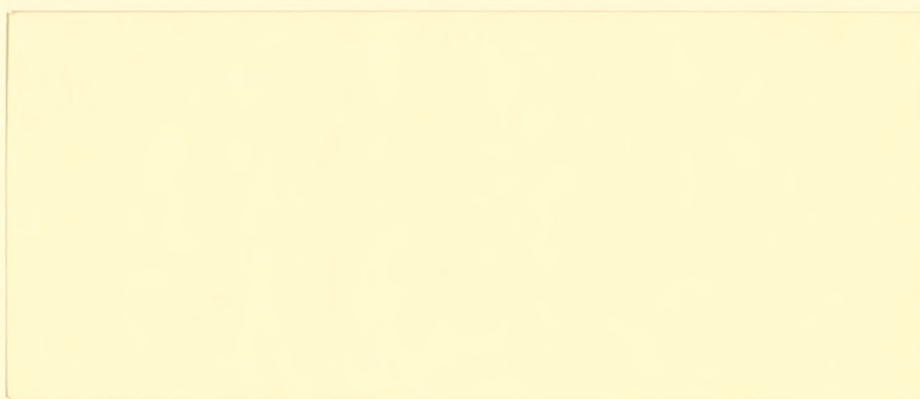
Tarek K. Abdel-Hamid

Stuart E. Madnick

February 1988

#WP 1980-88

MASSACHUSETTS
INSTITUTE OF TECHNOLOGY
50 MEMORIAL DRIVE
CAMBRIDGE, MASSACHUSETTS 02139



MODELING THE DYNAMICS OF
SOFTWARE PROJECT MANAGEMENT

Tarek K. Abdel-Hamid

Stuart E. Madnick

February 1988

#WP 1980-88

WORKING THE SYSTEM
SYSTEMATIC METHOD

John F. Abbot-Head

Street 2, Boston

1914-15

1914-15

MODELING THE DYNAMICS OF SOFTWARE PROJECT MANAGEMENT

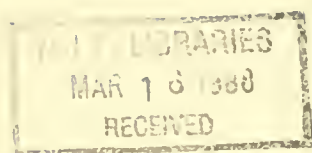
By

Tarek K. Abdel-Hamid
Naval Postgraduate School

Stuart E. Madnick
Massachusetts Institute of Technology

January, 1988

Submitted for publication to the Communications of the ACM



Abstract

The development of software systems has been marked by cost overruns, late deliveries, poor reliability, and users' dissatisfaction. The problems persist in spite of significant advances in the software engineering field to tackle the technological hurdles of software production. In recent years, the managerial aspects of software development have been gaining increasing recognition as being at the cores of both the problem and the solution.

The objective of this paper is twofold. First, we present a research paradigm for the study of software project management that is grounded in the feedback systems principles of system dynamics. Feedback processes are universal in social systems in general, and we show how, when applied to software project management, they do provide a powerful lens to view and understand software project behavior.

Second, we summarize the stream of research findings obtained to date. A system dynamics model of software project management has been developed and is being used to study and predict the dynamic implications of an array of managerial policies and procedures pertaining to the management of software development.

INTRODUCTION

The impressive improvements that are continuously being made in the cost-effectiveness of computer hardware are causing an enormous expansion in the number of applications for which computing is becoming a feasible and economical solution. This, in turn, is placing greater and greater demands for the development and operation of computer software systems. A conservative estimate indicates a "tenfold increase in the demand for software each decade, or a hundred fold increase between 1965 and 1985" (Musa, 1985).

The growth of the software industry has not, however, been painless. The record shows that the development of software has been marked by cost overruns, late deliveries, poor reliability, and users' dissatisfaction [(Buckley and Poston, 1984), (Ramamoorthy et al., 1984), and (Newport, 1986)].

In an effort to bring discipline to the development of software systems,

attempts have been made since the early 1970s to apply the rigors of science and engineering to the software production process. This lead to significant advances in the technology of software production (e.g., structured programming, structured design, formal verification, language design for more reliable coding, diagnostic compilers, and so forth).

The managerial aspects of software development, on the other hand, have attracted much less attention from the research community [(Thayer et al., 1981), (Zmud, 1980), and (Beck and Perkins, 1983)]. Cooper (1978) provides an insightful explanation for the reasons why:

Perhaps this is so because computer scientists believe that management per se is not their business, and the management professionals assume that it is the computer scientists' responsibility.

This "deficiency" in the field's research repertoire is being blamed by a growing number of researchers and practitioners for the persistence of the difficulties in producing software systems [(Pooch and Gehring, 1980), (Thomsett, 1980), and (Weinberg, 1982)]. A chief concern expressed is that, as of yet, we still lack a fundamental understanding of the software development process. And without such an understanding the possibility or likelihood of any significant gains on the managerial front is questionable [(Basili, 1982) and (McKeen, 1983)].

This paper reports on a stream of research designed to address the above concerns. Specifically, our goal is to develop a comprehensive system dynamics model of the software development process that enhances our understanding of, provide insight into, and make predictions about the process by which software development is managed. The following examples illustrate some of the critical management decisions that have been addressed in this research effort:

1. A project is behind schedule. Possible management actions include: revise completion date, hold to planned completion date but hire more staff, hold to planned completion date but work current staff overtime,

etc. What are the implications of these alternatives?

2. How much effort should be expended on quality assurance and how does that affect completion time and total cost?

3. The impact of different effort distributions among project phases (e.g., should the division of effort between development and testing be 80:20 or 60:40 percent?)

4. The reasons for and implications of the differences between potential productivity, actual productivity, and perceived productivity.

5. Why does the "90% syndrome" chronically recur?

In the remaining parts of this paper we present and discuss the integrative dynamic model of software project management that has been developed. We will provide an overview of both the model's structure and its behavior followed by a discussion of the insights gained. We begin our presentation, however, by first presenting arguments for the utility of the system dynamics modeling approach in the study of software project management.

THE HIGH COMPLEXITY OF THE SOFTWARE PROJECT MANAGEMENT PROCESS

Project management is often based on the simplistic "mental picture" captured by the single-loop model shown in Figure 1 (Roberts, 1981). The model portrays how project work is accomplished through the utilization of (1) project resources (manpower, facilities, equipment). As (2) work is accomplished on the project, it is reported (3) through some project control system. Such reports cumulate and are processed to create the (4) project's forecast completion time by adding to the current date the indicated time remaining on the job. Assessing the job's remaining time involves figuring out the magnitude of the effort (e.g., in man-days) believed by management to be remaining to complete the project, the level of manpower working on the

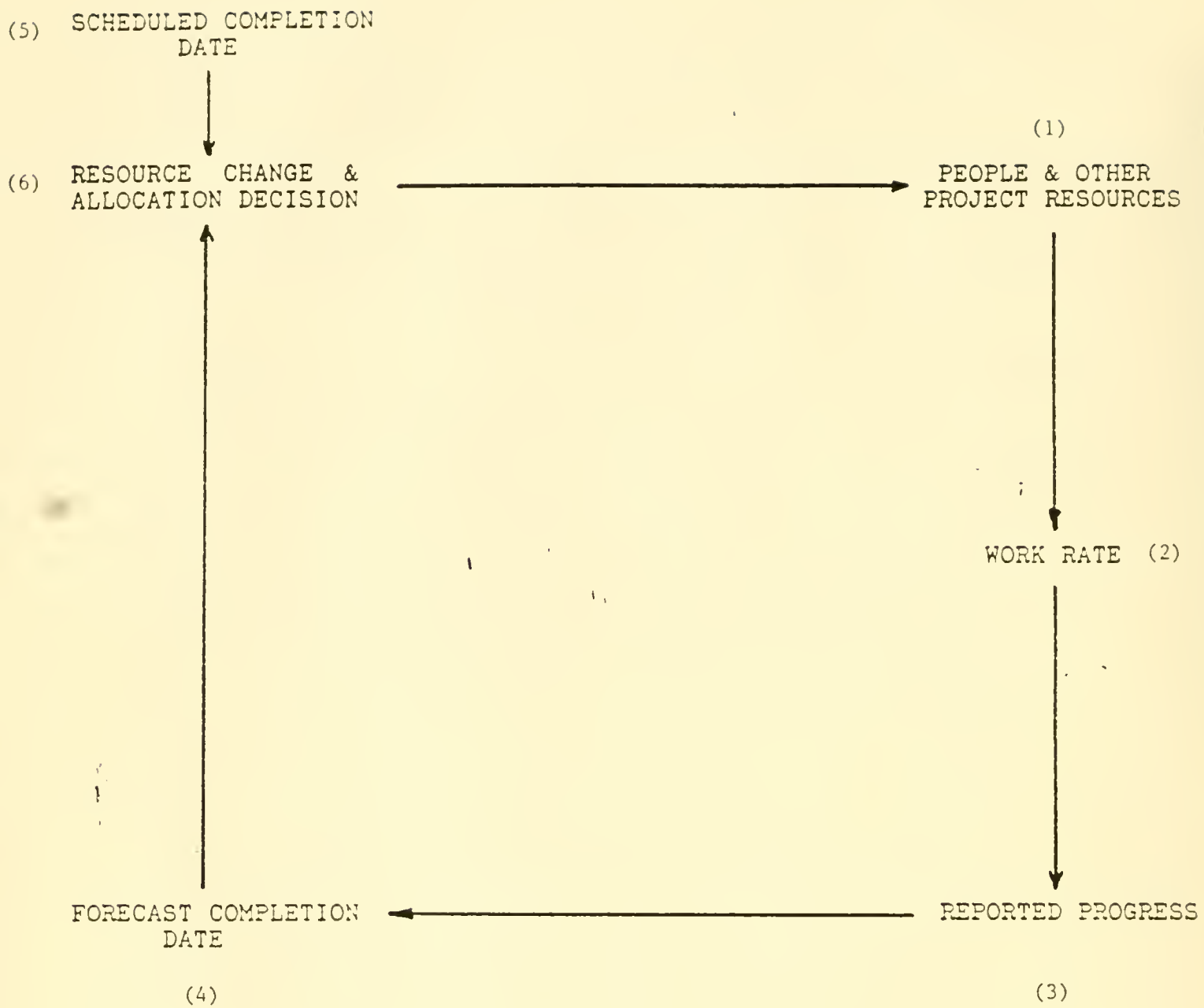


Figure (1)

A MODEL OF SOFTWARE PROJECT MANAGEMENT

5

project, and the perceived productivity of the project team. The feedback loop is completed (closed) as the difference, if any, between the (5) scheduled completion date and the (4) forecast completion date causes adjustments (6) in the magnitude or allocation of the project's resources.

What is attractive about the above model is that it is reasonable, simple, and manageable. But is it an adequate model of the dynamics of software project management?

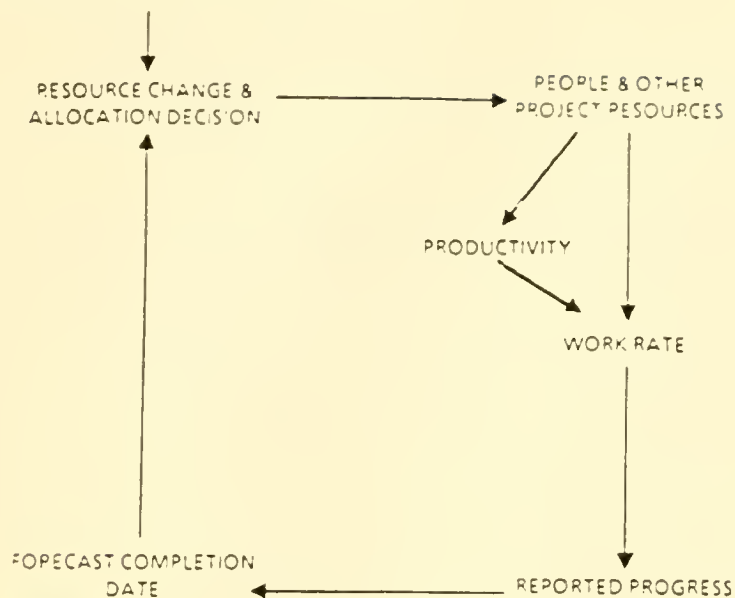
The software project management system is a far more complex conglomerate of interdependent variables that are interrelated in various nonlinear fashions. By excluding vital aspects of the real software project environment, the above model could seriously misguide the unsuspecting software manager. To see how, let us consider just a few of the many typical decisions pondered in a software project environment.

Adding more people to a late project: The mental picture of Figure 1 suggests a direct relationship between adding people resources and increasing the rate of work on the project, i.e., the higher the level of project resources the higher the work rate. This ignores one vital aspect of software project dynamics, namely, that adding more people often leads to higher communication and training overheads on the project, which can in turn dilute the project team's productivity. Lower productivity translates into lower progress rates, which can, therefore, delay the late project even further. This, in turn, can trigger an additional round of workforce additions and another pass around this "vicious cycle." These dynamic forces create the phenomenon often referred to as "Brooks' Law", i.e., that adding more people to a late software project makes it later (Brooks, 1978).

In Figure 2a we, therefore, amend Figure 1 by incorporating the vital link between the workforce level and productivity.

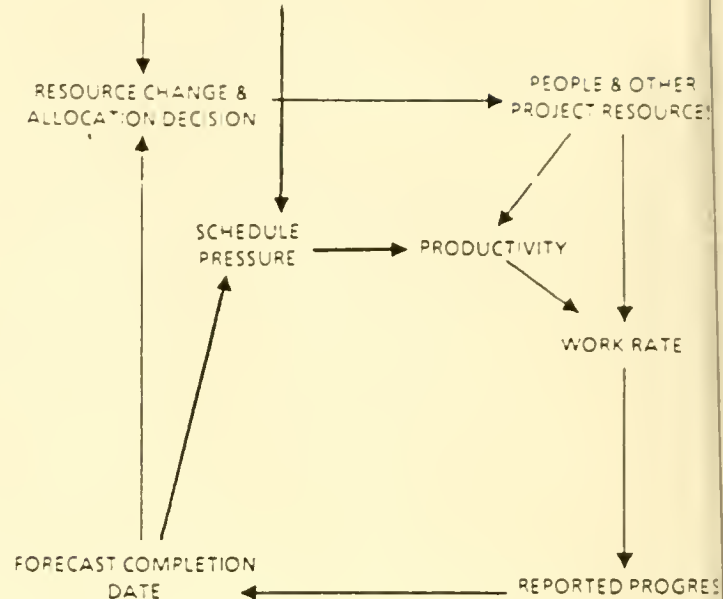
Adjusting the Schedule of a Late Project: Another part of the real system that is ignored by Figure 1 concerns the impact of intangible project

SCHEDULED COMPLETION DATE



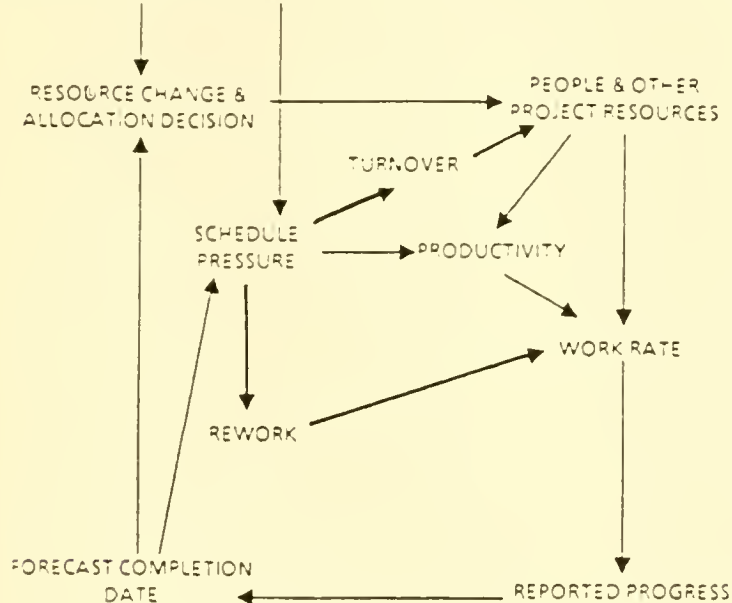
(a)

SCHEDULED COMPLETION DATE



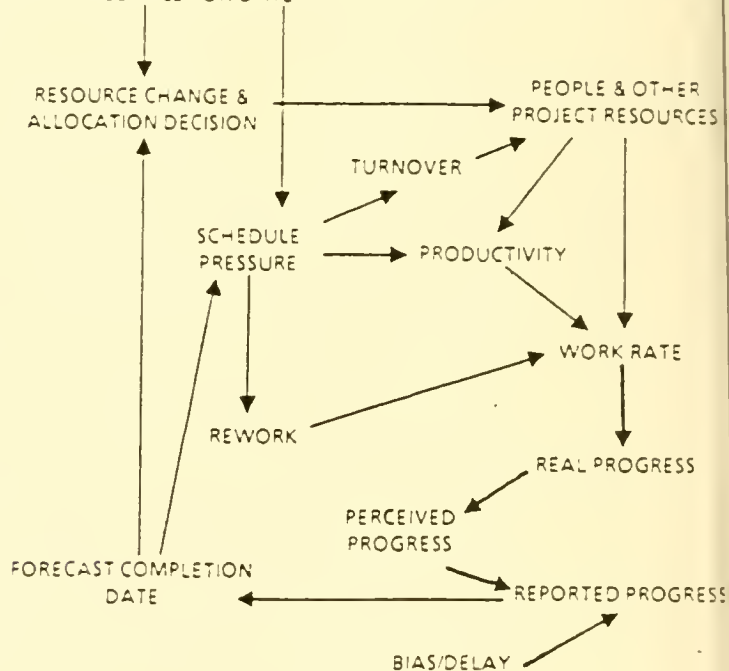
(b)

SCHEDULED COMPLETION DATE



(c)

SCHEDULED COMPLETION DATE



(d)

Figure (2)

AMENDMENTS TO THE PROJECT MANAGEMENT MODEL

pressures (e.g., schedule pressures) on the software developers' actions and decisions. For example, when faced with schedule pressures that arise as a project falls behind schedule, software developers typically respond by putting in longer hours and by concentrating more on the essential tasks of the job (Ibrahim, 1978). In one experiment, Boehm (1981) found that the number of man-hours devoted to project work increased by as much as 100%. This additional link between schedule pressure and productivity is captured in Figure 2b.

The impact of schedule pressures on software development is not limited to the above relatively direct role. Schedule pressures can also play less visible roles. For example, as Figure 2c suggests, schedule pressures can increase the error rate of the project team and thus the amount of rework on the project [(Radice, 1982) and (Mills, 1983)].

People under time pressure don't work better, they just work faster ... In the struggle to deliver any software at all, the first casualty has been consideration of the quality of the software delivered (DeMarco, 1982).

The rework necessary to correct such software errors obviously diverts the project team's effort from making progress on new project tasks, and thus can have a significant negative impact on the project's progress rate.

Also, consider the impact of schedule pressure on the workforce turnover rate (Figure 2c). There is evidence to suggest that workforce turnover increases when scheduling pressures persist in an organization (Freedland, 1987). This can be quite costly, since a higher turnover rate translates into lower productivity on the project.

Finally, How Really Late is a Late Software Project? Because software remains largely intangible during most of the development process (Baber, 1982), it is often difficult for project managers to assess real progress on the project. To the extent that the perceived progress rate differs from the real progress rate, an error in perceived cumulative progress will gradually

accumulate (Figure 2d). Furthermore, bias, often in the form of overoptimism, and delay in gathering and processing control information additionally distorts the reported progress. This undoubtedly poses yet another complication that is too real for the software project manager to exclude from a model of the process.

AN INTEGRATIVE SYSTEM DYNAMICS PERSPECTIVE OF SOFTWARE DEVELOPMENT

The above discussion illustrates that there are a large number of variables, both tangible and intangible, that impact the software development process. Furthermore, these variables are not independent, but are related to one another in complex fashions. Perhaps most importantly, understanding the behavior of such systems is complex far beyond the capacity of human intuition (Roberts, 1981).

A major deficiency in much of the research to date on software project management has been the inability to integrate our knowledge of the micro components of the software development process such as scheduling, productivity, and staffing to derive implications about the behavior of the total socio-technical system (Thayer, 1979). In the research effort described in this paper we build upon and extend what has been learnt about the micro components, to construct a holistic model of the software development process. It integrates the multiple functions of software development, including both the management-type functions (e.g., planning, controlling, staffing) as well as the software production-type activities (e.g., designing, coding, reviewing, testing).

A second unique feature of our modeling approach is the use of the feedback principles of system dynamics to structure and clarify the complex web of dynamically interacting variables. Feedback is the process in which an action taken by a person or thing will eventually affect that person or thing. Examples of such feedback systems in the software project environment

9
have already been demonstrated in the above discussion and are evident in Figures 1 and 2.

The significance and applicability of the feedback systems concept to managerial systems has been substantiated by a large number of studies (Roberts, 1981). For example, Weick (1979) observes that,

The cause-effect relationships that exist in organizations are dense and often circular. Sometimes these causal circuits cancel the influences of one variable on another, and sometimes they amplify the effects of one variable on another. It is the network of causal relationships that impose many of the controls in organizations and that stabilize or disrupt the organization. It is the patterns of these causal links that account for much of what happens in organizations. Though not directly visible, these causal patterns account for more of what happens in organizations than do some of the more visible elements such as machinery, timeclocks, ...

One of the pioneering works in the field is Roberts' (1964) published doctoral dissertation, which involved the development of a comprehensive system dynamics model of R&D project management. The model traces the full lifecycle of a single R&D project, and incorporates the interactions between the R&D product, the firm, and the customer. Roberts' work spurred a large number of system dynamics studies of project management phenomena. For example, Nay (1965) and Kelly (1970) extended Roberts' work in their research on multi-project environments. Richardson (1982) took a different tack, focusing on the development group. His model reproduces the dynamics of a development group over an eight year period as a continuous stream of products are developed and placed into production.

While the bulk of the system dynamics modeling work in the project management area has been devoted to the R&D environment, the applicability of the methodology to the domain of software production has been alluded to in the literature [e.g., (Graham, 1982), (Lehman, 1978), (Putnam, 1980), and (Snyder and Cox, 1985)]. Perhaps this should come as no surprise, since "the stages of research and development are similar in many respects to the stages of software analysis and design (Gehring and Pooch, 1977).

In the remaining sections of this paper we demonstrate how the system dynamics modeling technique was extended to the software project domain.

MODEL STRUCTURE AND BEHAVIOR

Before describing the model and experiments performed, there are several points that are important to clarify. First, due to its length and complexity only a portion of the entire model can be presented and explained in this paper. For more details the reader is referred to [(Abdel-Hamid, 1984) and (Abdel-Hamid and Madnick, 1988)]. Second, the focus of this research is on the dynamics of software projects; that is, aspects that change during the life of the project, such as workforce level and productivity, rather than aspects that are decided once and usually remain constant throughout the project, such as choice of programming language. Third, it is necessary to have a perspective about what the model is and is not intended to accomplish. This is particularly relevant because this research, as is most system dynamics work, is primarily intended to provide understanding of the dynamic behavior of a project (e.g., how variables like workforce-level and productivity change over time and why) rather than to provide point-predictions (e.g., of the number of errors generated).

The model was developed on the basis of field interviews of software project managers in five organizations. This was complemented by an extensive database of empirical findings from the literature. Figure 3 depicts the model's four subsystems, namely: (1) the Human Resource Management Subsystem; (2) the Software Production Subsystem; (3) the Controlling Subsystem; and (4) the Planning Subsystem. The figure also illustrates some of the interrelationships among the four subsystems.

THE HUMAN RESOURCE MANAGEMENT SUBSYSTEM

The Human Resource Management Subsystem, shown in Figure 4, captures the

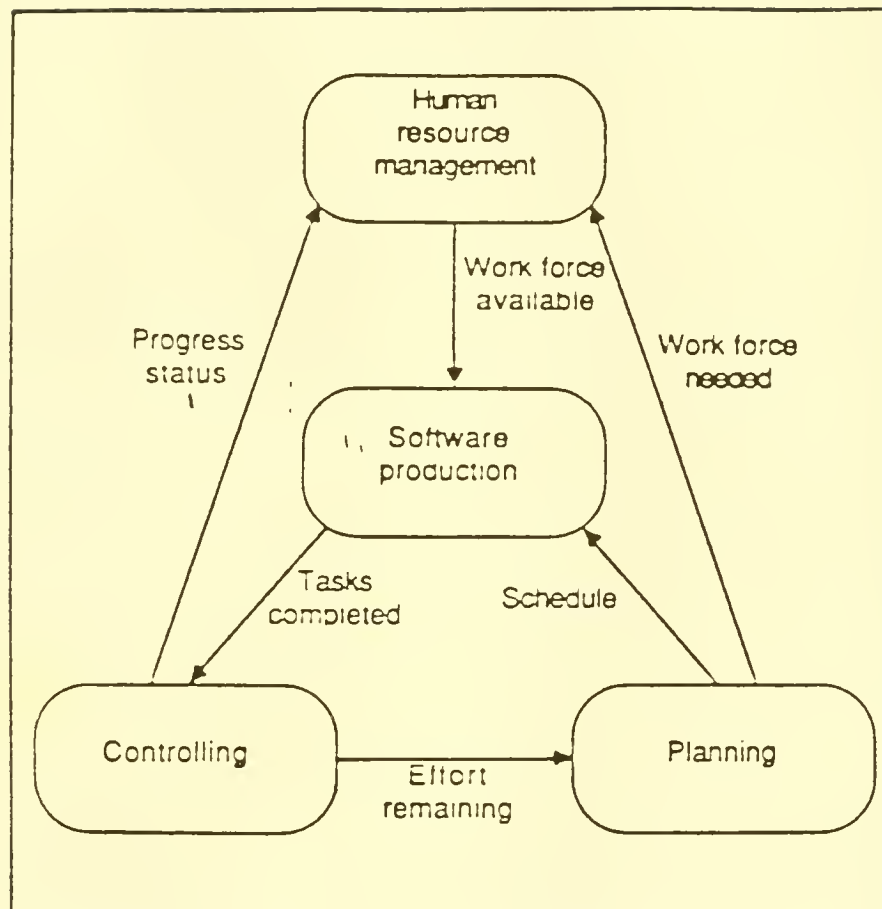


Figure (3)

THE SYSTEM DYNAMICS MODEL'S FOUR SUBSYSTEMS

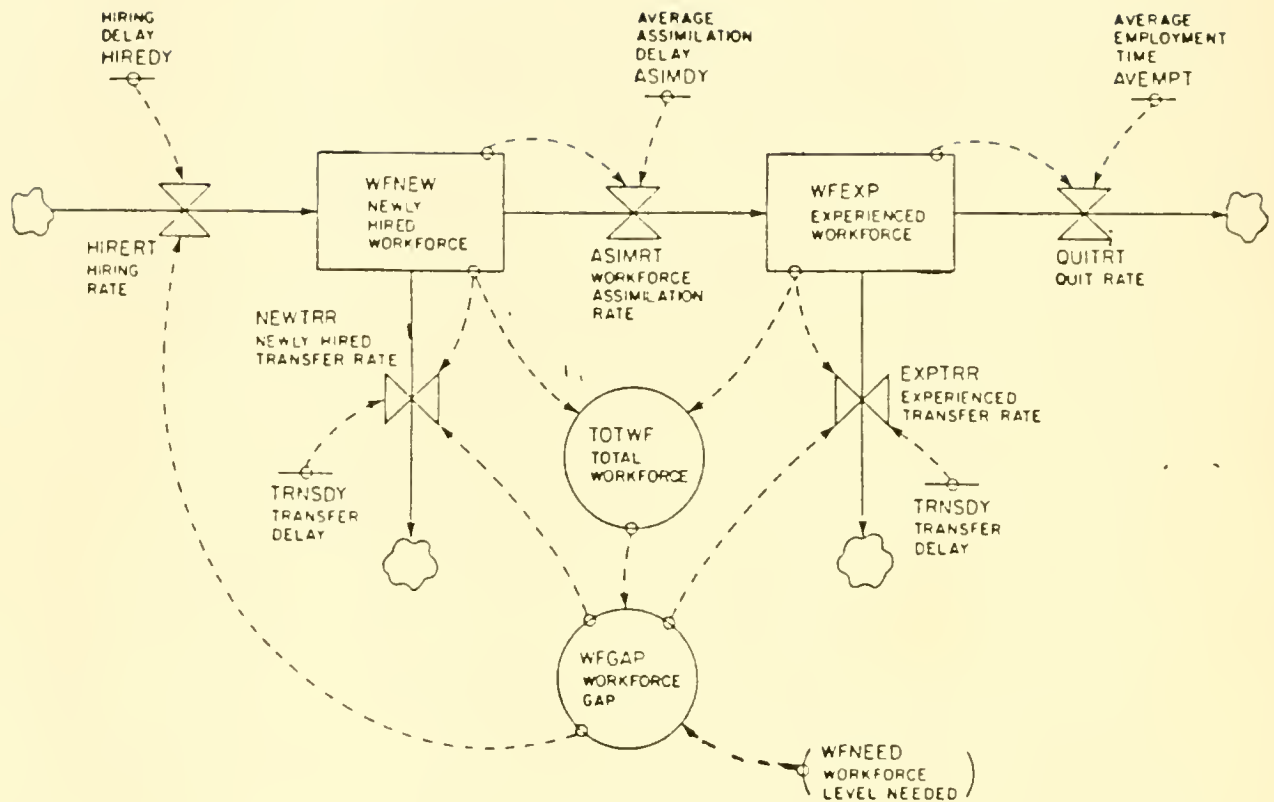
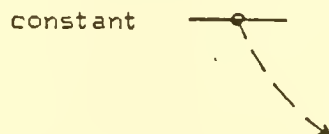


Figure (4)

THE HUMAN RESOURCE MANAGEMENT SUBSYSTEM

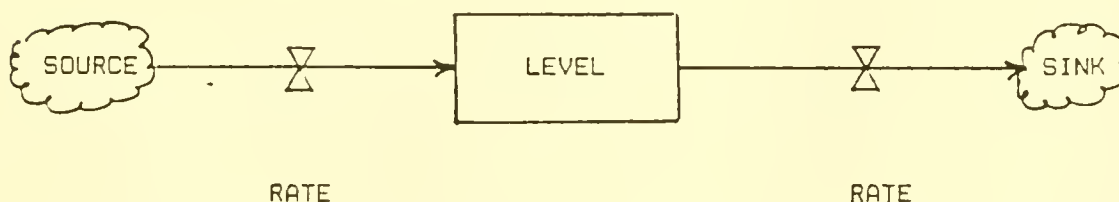
hiring, training, assimilation, and transfer of the human resource.

The schematic conventions used in Figure 4 are the standard conventions used in system dynamics models. All the quantities appearing in such models can be classified in two broad groups: constants, whose values cannot change at all in the course of a simulation, and variables, whose values can. The symbol for a constant is shown below:



Model variables are one of three types, namely, "level," "rate," and "auxiliary" variables. A level is an accumulation, or an integration, over time of flows or changes that come into and go out of the level. The flows increasing and decreasing a level are called rates. Thus, "NEWLY HIRED WORKFORCE" is a level of people that is increased by the "HIRING RATE" and decreased by the "WORKFORCE ASSIMILATION RATE."

Rates and levels are represented as shown below:



The cloud-like symbols represent sources and sinks for the "stuff" that flows into and out of levels. For example, for a level of workforce these cloudlike symbols represent where people come from when they are hired and where they go after leaving the project. Their presence indicate that the real-world accumulations they represent lie outside the boundary of the

system being modeled.

The flows that are controlled by the rates are either information flows or physical flows. We will use the two types of arrow designators shown below:

INFORMATION FLOWS ----->

OTHER FLOWS 
(e.g., People)

Levels and rates are sufficient, in principle, to represent all variables in a system dynamics model. Usually, however, it is very difficult to write a rate equation without first doing some (often complex) algebraic computations. These additional algebraic computations are termed "auxiliaries." Thus, auxiliary variables, as their name implies, aid in the formulation of rate equations. For example, the "HIRING RATE" equals the "WORKFORCE GAP" (an auxiliary which is equal to the difference between the "WORKFORCE LEVEL NEEDED" and the current "TOTAL WORKFORCE") divided by the "HIRING DELAY". Auxiliary variables are represented by a circular symbol.



Finally, variables that are defined in sectors of the model other than the one(s) diagrammed are represented by enclosing the variable name in parentheses as shown below.

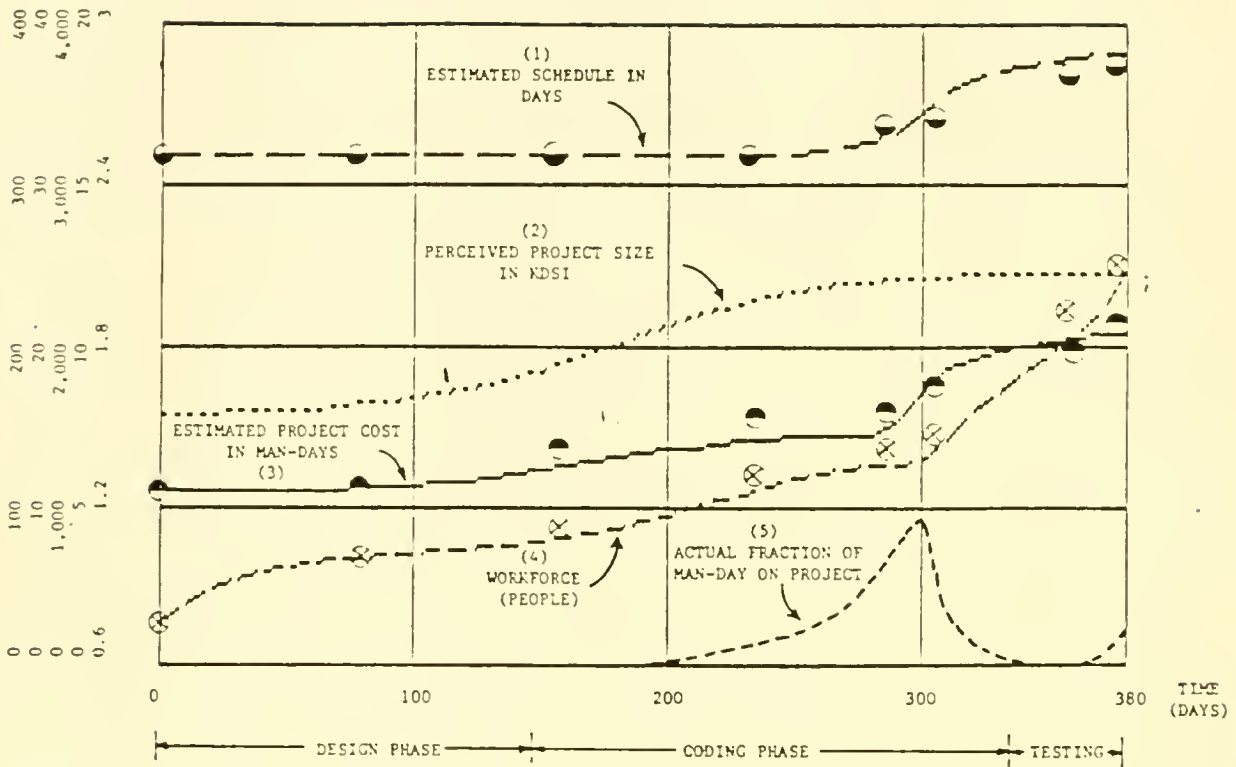
(VARIABLE FROM
ANOTHER SECTOR)

Returning to Figure 4, notice that the project's total workforce is comprised of two workforce levels, namely, "Newly Hired Workforce" and "Experienced Workforce." (In the current version of the model, workforce is more finely divided into four, not two, levels. However, for this presentation the simpler structure of Figure 4 is quite sufficient to demonstrate the basic ideas.) Segregating the workforce into these two categories of employees is necessary for two reasons. First, newly added team members are less productive (on the average) than the "old timers" (Cougar and Zawacki, 1980). Secondly, it allows us to capture the training processes involved in assimilating the new members into the project team.

On deciding upon the total workforce level desired, project managers consider a number of factors. One important factor, of course, is the project's completion date. As part of the planning subsystem (to be discussed later), management determines the workforce level that it believes is necessary to complete the project on time. In addition, though, consideration is also given to the stability of the workforce. Thus, before adding new project members, management contemplates the duration for which the new members will be needed. In general, the relative weights given to workforce stability versus on-time completion change with the stage of project completion. For example, toward the end of the project there could be considerable reluctance to bring in new people. This reluctance arises from the realization that there just wouldn't be enough time to acquaint the new people with the mechanics of the project, integrate them into the project team, and train them in the necessary technical areas.

Figure 5 demonstrates the workforce staffing pattern in an example model output. It depicts the model's simulation of NASA's DE-A software project (NASA, 1983). The project was conducted at the Goddard Space Flight Center

curve (1)
curve (2)
curve (3)
curve (4)
curve (5)



- DE-A's actual "Estimated Schedule in Days"
- DE-A's actual "Estimated Project Cost in Man-Days"
- ⊗ DE-A's actual "Workforce" (full-time-equivalent people)

Figure (5)

MODEL SIMULATION OF THE DE-A PROJECT

(GSFC) to design, implement, and test a software system for processing telemetry data and providing attitude determination and control for the DE-A satellite. The project's size was 24,000 delivered source instruction (24 KDSI), the development and target operations machines were the IBM S/360-95 and -75, and the programming language was FORTRAN. Initially, the project was estimated to require 1,100 man-days and to be completed in 320 working days. The actual results were 2,200 man-days and 380 days, respectively.

The model's results conformed quite accurately to the project's actual behavior (represented by the O points in the figure). Notice that the project's workforce pattern differs from the "typical" pattern discussed in the literature, i.e., the concave type curve that rises, peaks, and then drops back to lower levels as the project proceeds towards the system testing phase (Boehm, 1981). Because NASA's launch of the DE-A satellite was tied to the completion of the DE-A software, serious schedule slippages were not tolerated. Specifically, all software was required to be accepted and frozen 90 days before launch. As this date was approached, pressures developed that overrode normal workforce stability considerations. That is, project management became increasingly willing to "pay any price" necessary to avoid overshooting the 90-day-before-launch date. This translated, as the figure indicates, into a management that was increasingly willing to add more people. [In (Abdel-Hamid, 1988a) we investigate whether such a staffing policy did or did not contribute to the project's late completion.]

THE SOFTWARE PRODUCTION SUBSYSTEM

This subsystem models the software development process. The operation and maintenance phases of the software lifecycle arc, thus, not included. The development lifecycle phases incorporated include the designing, coding, and testing phases. Notice that the initial requirements definition phase is also excluded. There are two reasons for this. The primary reason relates to the

desire to focus this study on the "endogenous" software development organization i.e., the project managers and the software development professionals, and how their policies, decisions, actions, etc. affect the success/failure of software development. The requirements definition phase was, thus, excluded since in many environments the definition of user requirements is not totally within the control of the software development group (McGowan and McHenry, 1980). Second, "Analysis to determine requirements is distinguished as an activity apart from software development. Technically, the product of analysis is non-procedural (i.e., the focus is functional)" (McGowan and McHenry, 1980).

An overview of the Software Production Subsystem is shown in Figure 6. As the software is developed, it is also reviewed to detect any errors e.g., using quality assurance activities such as structured walkthroughs. Errors detected through such activities are reworked. Not all software errors are detected during development, however, some "escape" detection until the testing phase.

One of the subsystem's more interesting components is that of software productivity, which is expanded in Figure 7. Its structure is based on the work of the psychologist Ivan Steiner (1972). Steiner's model can simply be stated as follows:

$$\text{Productivity} = \text{Potential Productivity} - \text{Losses Due to Faulty Process}$$
where losses due to faulty process refer to a group's communication and motivation losses.

Potential Productivity: According to Steiner, potential productivity is defined as "the maximum level of productivity that can occur when an individual or group ... makes the best possible use of its resources". It is a function of two sets of factors, the nature of the task (e.g., product complexity, database size) and the group's resources (e.g., personnel capabilities, software tools). The effects of such factors on software

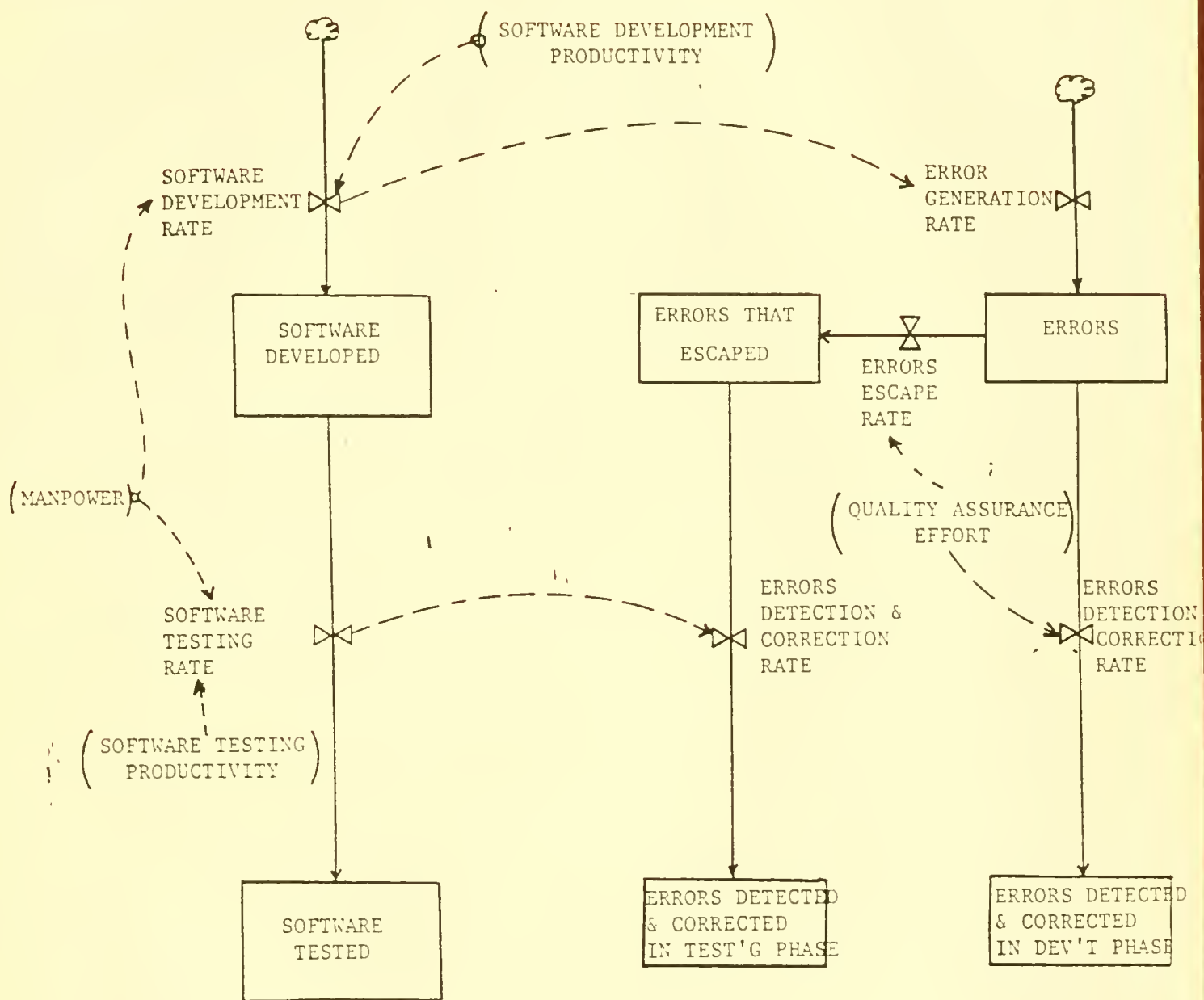
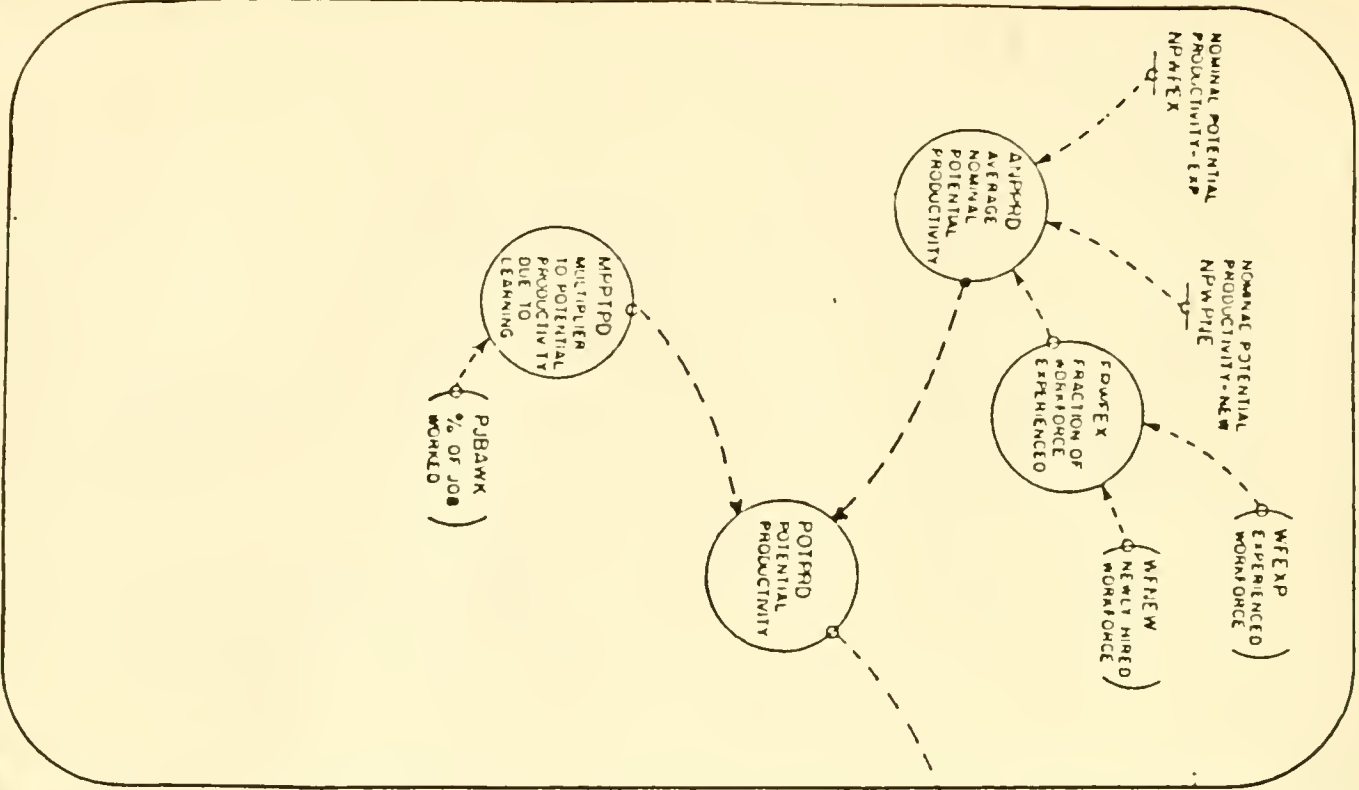


Figure (6)
THE SOFTWARE PRODUCTION SUBSYSTEM

Potential Productivity



Communication & Motivation Losses

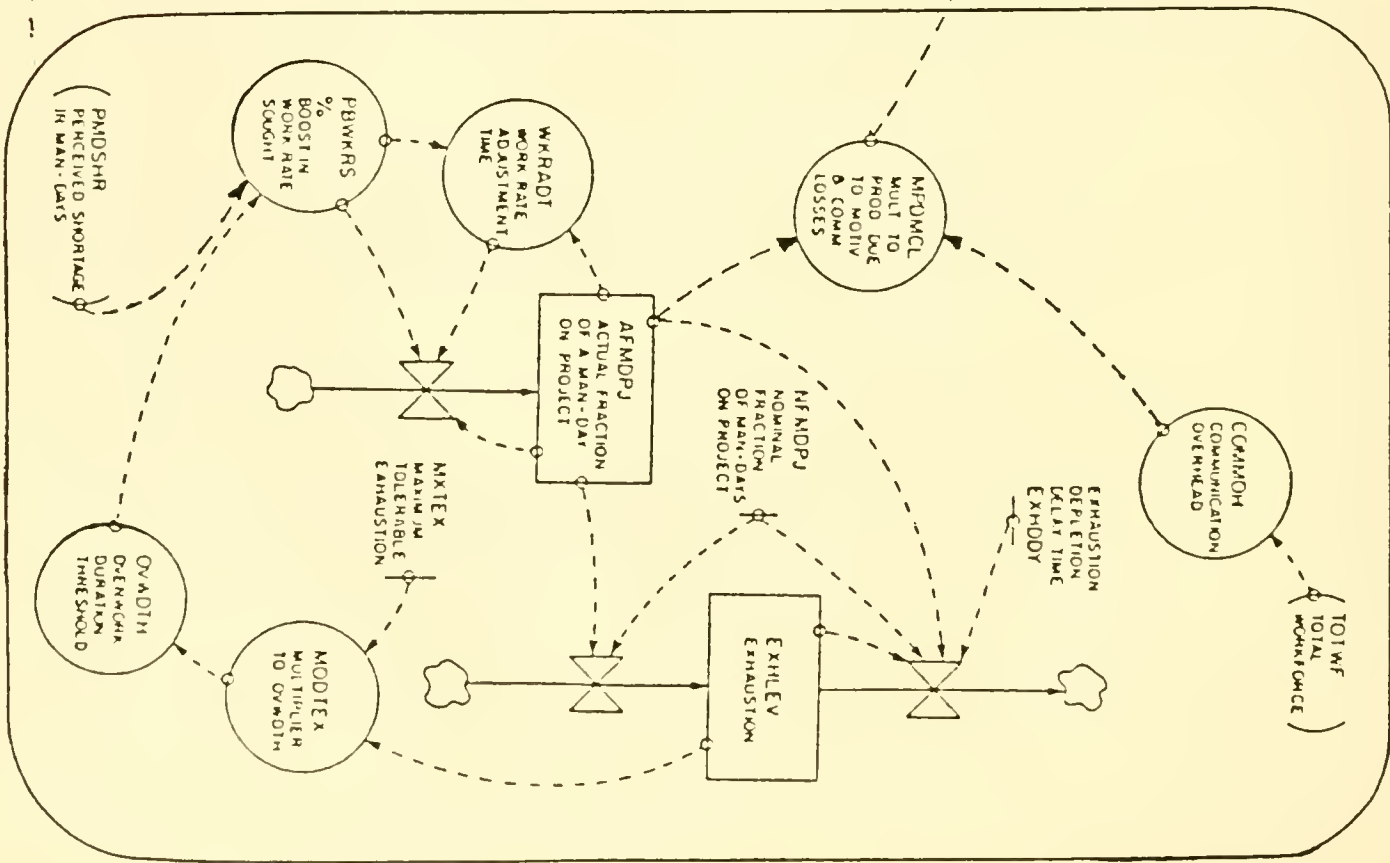


Figure (7)

productivity have been widely investigated in the literature [e.g., see (Scott and Simmons, 1974), (Chrysler, 1978), and (Boehm, 1981)].

Notice that while most of the above factors do vary from organization to organization (e.g., availability of software tools, personnel capability) and from project to project within a single organization (e.g., product complexity, database size) they, however, tend to remain constant throughout the development lifecycle of any single project. This means that in studying the dynamics of software productivity during the lifecycle of a particular software project, which is our concern here, the above variables can be assumed to remain constant. In the model, such factors are captured through the model's nominal potential productivity parameters.

The "Average Nominal Potential Productivity" for the workforce as a whole is the weighted average of two nominal potential productivity parameters, one to represent that of the average experienced staff member, and the second represents that of the average newly hired employee. They represent the maximum level of software development productivity that can be attained for a specific project situation (i.e., a specific project within a specific organization) by the respective employee type (whether experienced or new), at some point in the project.

The "Average Nominal Potential Productivity" is not a constant, it changes dynamically throughout the lifecycle as the workforce experience mix changes, and as the workforce learns from working on the project (Weinberg, 1982).

Losses Due to Faulty Processes: Actual productivity rarely equals potential productivity due to losses caused by communication and motivation overheads. The effects of communication and motivation losses have a multiplicative effect. Motivation factors first influence the fraction of a man-day devoted to project work. Since time is often lost on personal matters, coffee-breaks, and other miscellaneous non-project related

activities, this fraction will usually have a value less than 1.

Communication losses, then, take further "cuts," as we explain below.

Motivation-Type Losses: To discern the dynamic effects of motivation losses on productivity we need (again) to distinguish between those factors that tend to remain constant during the life of any one particular project and those that change throughout the lifecycle. Many of the motivational factors discussed in the literature (e.g., possibility for growth and advancement, level of responsibility, and salary) are factors that characterize the overall organizational setting and climate. In our formulation, such non-dynamic factors are implicitly incorporated within the definition of the potential productivity parameters.

"Another motivation approach which is particularly appropriate to the data processing area is goal setting" (Bartol and Martin, 1982). The authors suggest that project goals and schedules play an important motivational role throughout the life of a software project. This was corroborated by Boehm (1981), who found that schedule pressures and project deadlines can significantly expand/contract the project members' "slack time" ... which is the time lost on off-project activities such as coffee-breaks, personal business, non-project communication.

The "Nominal Fraction of a Man-Day on Project" represents the fraction of daily hours allocated to project-related work in the absence of schedule pressures. In designating a value for this parameter one can draw upon a large volume of research findings. The findings are clustered within the 50-70% range [e.g., 50% (Brooks, 1978), 50-60% (Pooch and Gehring, 1980), 60% (Thadhani, 1984), and 70% (Boehm, 1981)]. On the basis of these findings, the value of "Nominal Fraction of a Man-Day on Project" is set in the model to 0.60. That is, in an 8-hour day, a full-time employee allocates, on the average, only $0.6 \times 8 = 4.8$ hours to the project.

This loss in productivity does not, of course, remain constant throughout the life of the project. Schedule pressures can push the "Actual Fraction of a

Man-Day on Project" to both higher values (under positive schedule pressure) as well as to lower values (under negative schedule pressure).

Positive schedule pressures can arise in a project whenever the project is perceived to be behind schedule. When confronted with such a situation, software developers tend to work harder by allocating more man-hours to the project in an attempt to compensate for the perceived "delinquency" and bring the project back on schedule [(DePree, 1984) and (Ibrahim, 1978)].

But what if such a situation persists ... would workers be willing to work harder indefinitely? The answer, based on common sense and confirmed by our field study results was overwhelmingly no. Our findings indicate that there are limits for how long employees are willing to work at an above-normal rate. Workers savor their slack time (e.g., coffee breaks, social communications, personal business), and they typically will not tolerate a prolonged deprivation of such "breathers." A compressed slack time, therefore, exhausts them (psychologically more so than physically) in the sense that it cuts into their tolerance level for continued over-working. (A significant portion of the detailed productivity structure in the model is devoted to handling these intangible dynamic forces.)

Negative schedule pressures, on the other hand, arise on those (probably rare) occasions when a project is perceived to be ahead of schedule. This happens, for example, when management initially over-estimates the project's scope. When project members perceive some excesses in the schedule, parts, if not all, of these excesses tend to get "absorbed" by the workers in the form of goldplating and/or "under-work" [(DePree, 1984) and (Ibrahim, 1978)]. For example:

... if the software cost or schedule estimate for meeting a milestone is higher than the ideal, Parkinson's Law indicates that people will use the extra time for ... personal activities, catching up on the mail, etc (Boehm, 1981).

Analogous to the case of positive schedule pressure, there are limits on

how much "fat" employees would be willing, or allowed, to absorb. Beyond these limits, excesses are translated into cuts in the project's schedule.

Losses Due to Communication Overhead: The value of the "Actual Fraction of a Man-Day on Project" captures the losses in productivity due to motivational factors only. Additionally, losses in productivity due to communication overhead may be incurred. Communication overhead is the drop in productivity of the average team member caused by the losses incurred in communicating (verbally as well as non-verbally) with others on the project.

The nature of the relationship between communication overhead and team size has been investigated by several authors. It is widely held that communication overhead increases in proportion to n^2 , where n is the size of the team [(Brooks, 1978), (Mills, 1976), (Scott and Simmons, 1975), (Shooman, 1983), and (Zelkowitz, 1978)].

The dynamic behavior of the "ACTUAL FRACTION OF A MAN-DAY ON PROJECT" for the DE-A project is depicted in Figure 5. Notice the "spike" that occurs as the end-of-development milestone is approached. To understand the causes of this behavior, we need first to observe (also in Figure 5) that when the project started, its true size was underestimated by 35%. (The value of "Perceived Project Size" starts at 16 KDSI and eventually rises to 24.4 KDSI.) As the "new" job tasks were discovered, upward adjustments were made in the project's estimated man-days. As is typically the case, however, the man-day adjustments were not quite enough. This created a deficit in the project's budget which became visible only towards the end of the development phase when the development work was almost finished and the man-days close to used up.

As the man-day deficit became visible, the project team reacted by working harder and longer hours in an attempt to bring the project back on track. This translates in the model into the higher values of the "Actual Fraction of a Man-Day on Project." But as was explained above, project teams do not maintain an above-normal work rate indefinitely. The persistence of the work backlog,

thus, eventually overwhelms the workforce's intensified efforts, and around day 300 arrangements were made to adjustment both the project's man-day estimate and its schedule.

THE CONTROL SUBSYSTEM

Decisions made in any organizational setting are based on what information is actually available to the decision maker(s). Often, this available information is inaccurate. Apparent conditions may be far removed from the actual or true state, depending on the information flows that are being used and the amount of time lag and distortion in these information flows (Forrester, 1961).

True project progress is a good example of a project variable that is often difficult to assess (see Figure 8). Because software is basically an intangible product during most of the development process, "It is difficult to measure performance in programming ... It is difficult to evaluate the status of intermediate work such as undebugged programs or design specification and their potential value to the complete project " (Mills, 1983).

How, then, is progress measured in a software project? Our own field study findings corroborate those reported in the literature, namely, that progress, especially in the earlier phases of software development, is measured by the rate of expenditure of resources rather than by some count of accomplishments (DeMarco, 1982). Baber (1982) explains:

It is essentially impossible for the programmers to estimate the fraction of the program completed. What is 45% of a program? Worse yet, what is 45% of three programs? How is he to guess whether a program is 40% or 50% complete? The easiest way for the programmer to estimate such a figure is to divide the amount of time actually spent on the task to date by the time budgeted for that task. Only when the program is almost finished or when the allocated time budget is almost used up will he be able to recognize that the calculated figure is wrong.

When progress in the earlier phases of software development, is measured by the rate of expenditure of resources, status reporting ends up being

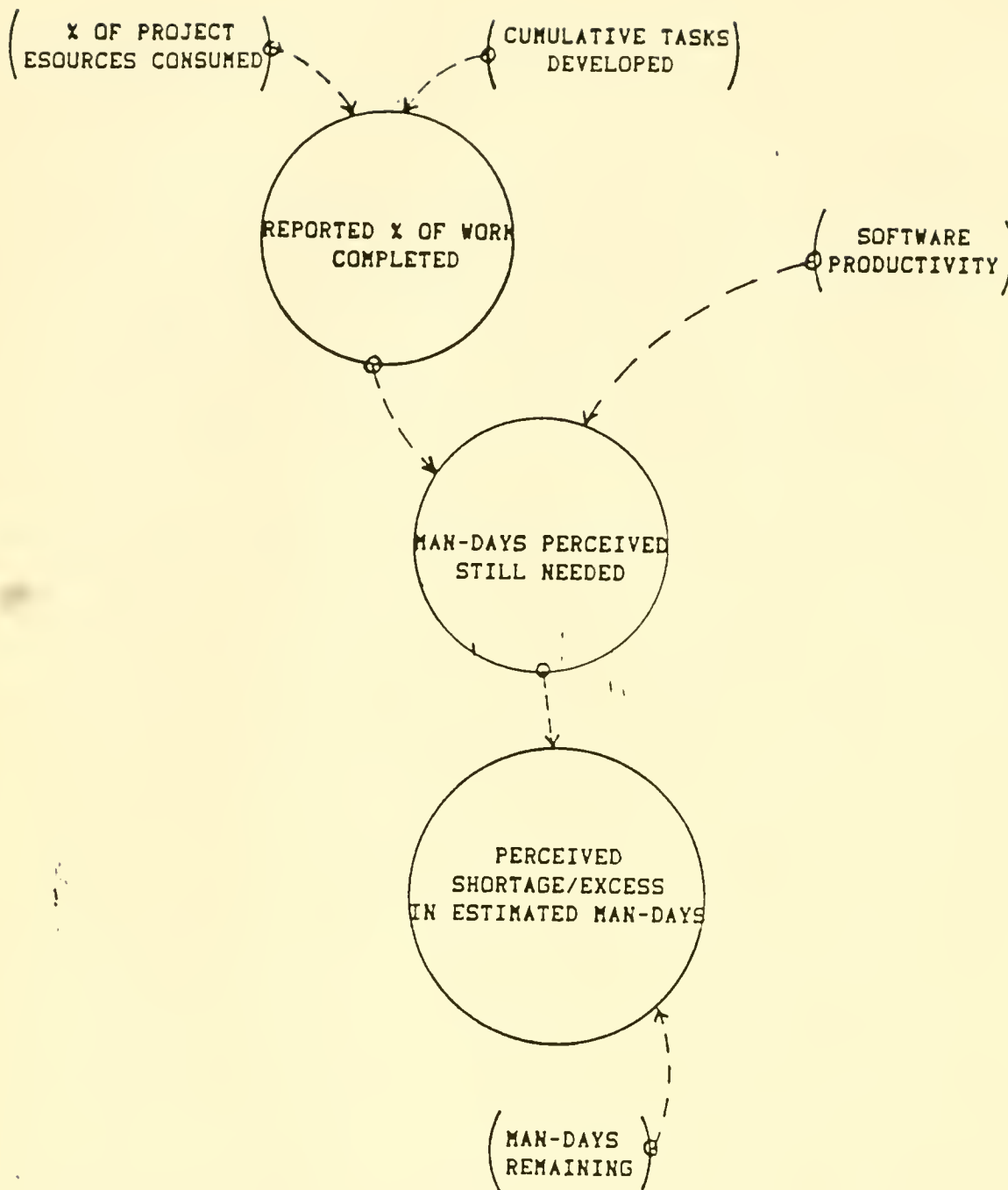


Figure (8)
CONTROLLING SUBSYSTEM

nothing more than an echo of the original plan. One "infamous" consequence of this is the "90% syndrome phenomenon." Baber (1982) provides the following description of the problem:

... estimates of the fraction of the work completed (increase) as originally planned until a level of about 80-90% is reached. the programmer's individual estimates then increase only very slowly until the task is actually completed.

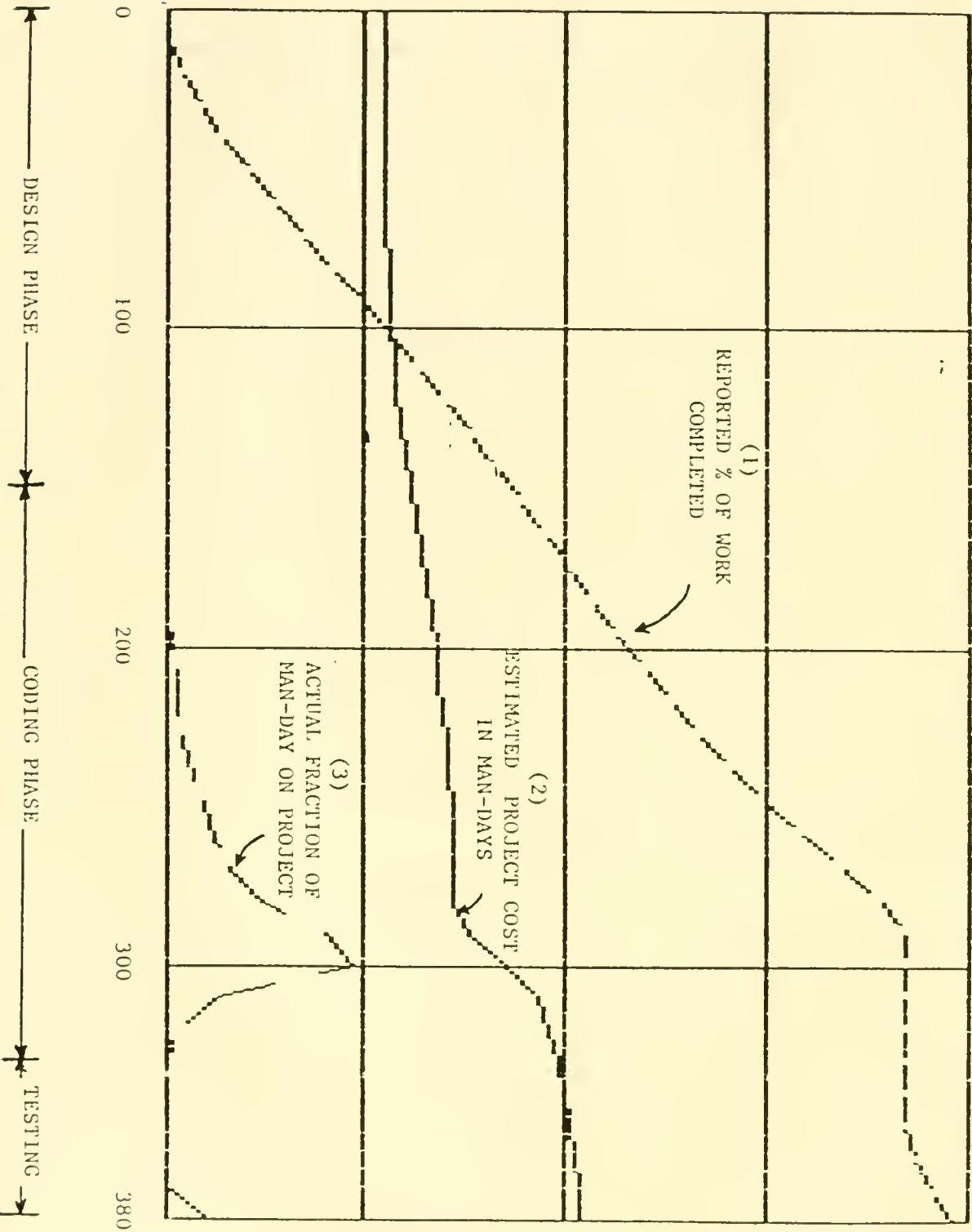
There is ample evidence in the literature on the pervasiveness of the 90% syndrome in software development projects (DeMarco, 1982). Its manifestation in the simulated DE-A project is depicted in Figure 9. By measuring progress in the earlier phases of the project by the rate of expenditure of resources, status reporting ended up being nothing more than an "illusion" that the project was right on target. However, as the project approached its final stages (e.g., when 80-90% of the resources are consumed), discrepancies between the percent of tasks accomplished and the percent of resources expended became increasingly more apparent. At the same time, project members became increasingly able to perceive how productive the workforce has actually been. This results in a better appreciation of the amount of effort actually remaining. As this appreciation developed, it started to, in effect, discount the project's progress rate. Thus, although the project members proceeded towards the final stages of the project at a high work rate because of schedule pressures, their net progress rate slowed down considerably. This continued until the project completed.

THE PLANNING SUBSYSTEM

In the Planning Subsystem, initial project estimates (e.g., for completion time, staffing, man-days) are made at the beginning of the project. These estimates are then revised, as necessary, throughout the project's life. For example, to handle a project that is perceived to be behind schedule, plans can be revised to add more people, extend the schedule, or do a little of both. The Planning Subsystem is depicted in Figure 10.

curve (1)
curve (2)
curve (3)

0	25	50	75	100
0	1,000	2,000	3,000	4,000
0.6	1.2	1.8	2.4	3



TIME
(DAYS)

Figure (9)
REPORTED % OF WORK COMPLETED

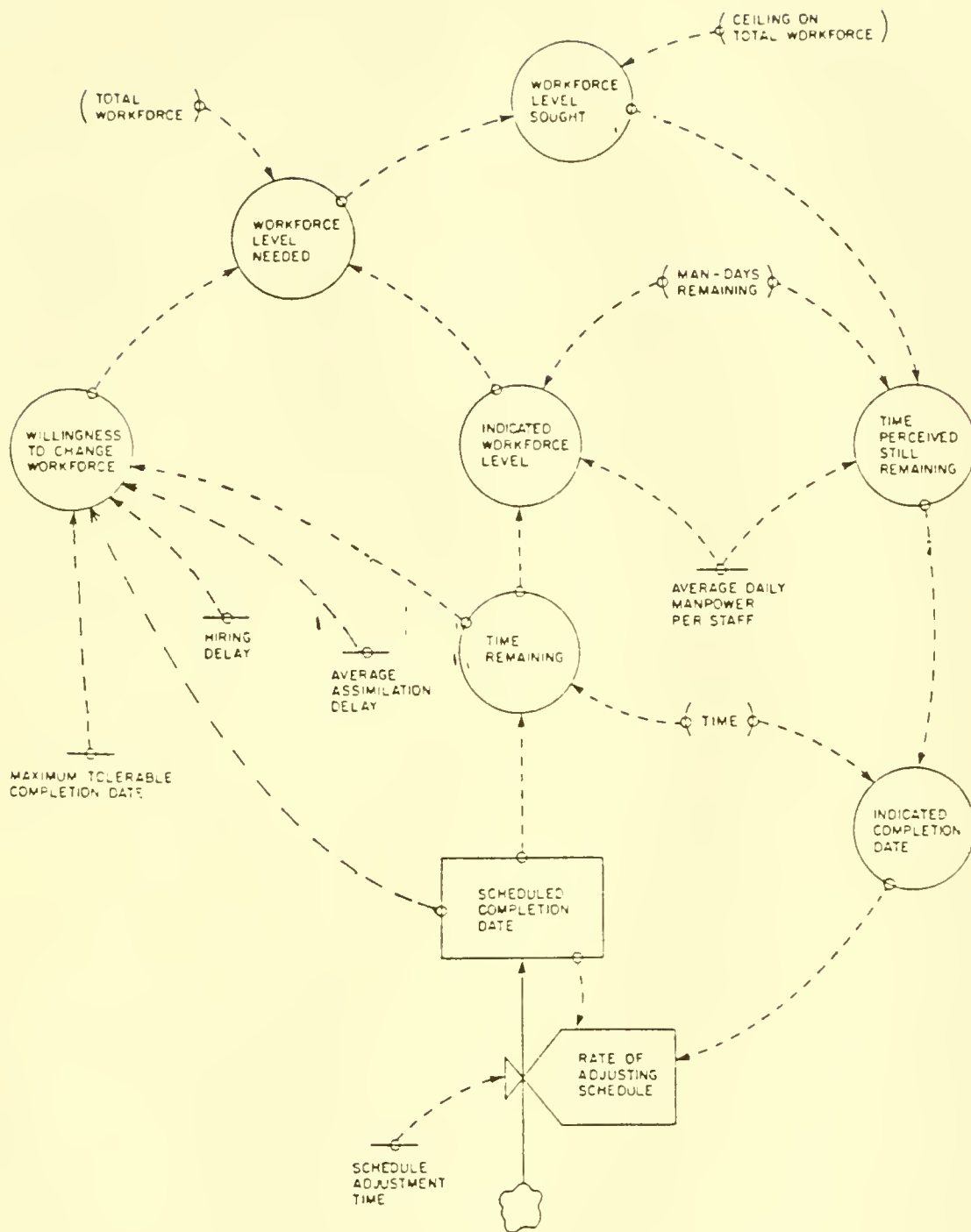


Figure (10)
PLANNING SUBSYSTEM

By dividing the value of "Man-Days Remaining" at any point in the project, by the "Time Remaining" a manager can determine the "Indicated Workforce Level." This represents the workforce size believed to be necessary and sufficient to complete the project on time. If this indicated workforce size turns out to be lower than the value of the actual workforce level on the project, excessive employees would simply be transferred out of the project (as indicated by the TRANSFER RATES of Figure 4). If on the other hand, the opposite is true, then this would indicate a need to add more people.

Hiring decisions are not determined, however, solely on the basis of scheduling considerations. Consideration is typically given to the stability of the workforce. Different organizations weigh this factor to various extents. For example, workforce stability had relatively little weight in NASA's DE-A project, as we saw. Such organizational differences are captured in the model by the policy variable termed "Willingness to Change Workforce Level," and whose form is organization-specific.

By dividing the value of the "Workforce Level Sought" (that emerges after the above set of factors is contemplated) into the value of the "Man-Days Remaining," management determines the time it perceives is required to complete the project. Once this, in turn, is known, it can be used to adjust the project's "Scheduled Completion Date," if necessary.

Referring back to Figure 5, notice how project DE-A's management was inclined not to adjust the project's scheduled completion date during most of the development phase of the project. Adjustments, in the earlier phases of the project, were instead made to the project's workforce level. This behavior is not atypical. It arises, according to DeMarco (1982) because of political reasons:

Once an original estimate is made, it's all too tempting to pass up subsequent opportunities to estimate by simply sticking with your previous numbers. This often happens even when you know your old estimates are substantially off. There are a few different possible explanations for this effect: It's too early to show slip ... If I re-estimate now, I risk having to do it again later (and looking bad twice) ... As you can see, all such

reasons are political in nature.

With this discussion of the model's Planning Subsystem we conclude our overview presentation of the model's structure and behavior.

MODEL VALIDATION

The process of judging the validity of a system dynamics model includes a number of objective tests (Richardson and Pugh, 1981)]. They include:

- Face validity. To test the fit between the rate/level/feedback structure of the model and the essential characteristics of the real system. This was confirmed by the software project managers involved in the study.
- Replication of reference modes. To test whether the model can endogenously reproduce the various reference behavior modes characterising the system under study. Reference modes include problematic behavior patterns such as the "90% syndrome," and observed responses to past policies such as DE-A's scheduling behavior, etc.
- Extreme condition simulations. To test whether the model behaves reasonably under extreme conditions or extreme policies. A model that does not behave reasonably under extreme conditions (e.g., hiring delay equals to zero) is suspect, because one may not be certain when aspects of extreme conditions may crop up in ordinary runs.
- Case study. The DE-A project case-study which was conducted after the model was completely developed, constituted an important element in validating model behavior. (NASA was not one of the five organizations studied during model development.) The fact that the model was able to replicate some unusual behavior e.g., in the staffing area, in addition to the more traditional workforce patterns observed in the five project environments studied during the development of the model, indicates its robustness.

"Any one of these test by itself is certainly inadequate as an indicator

of model validity. Taken together, (however), they are a formidable filter" (Richardson and Pugh, 1981).

EXPERIMENTS UNDERTAKEN AND IMPLICATIONS OF RESULTS

"In software engineering it is remarkably easy to propose hypotheses and remarkably difficult to test them" (Weiss, 1979). Many in the field have, thus, argued for the desirability of having a laboratory tool for testing ideas and hypotheses in software engineering (Thayer, 1979).

The computer simulation tools of system dynamics provide us with such an experimentation vehicle. The effects of different assumptions and environmental factors can be tested. In the model system, unlike the real systems, the effect of changing one factor can be observed while all other factors are held unchanged. Internally, the model provides complete control of the system's organizational structure, its policies, and its sensitivities to various events (Forrester, 1961).

Currently, the model is being used to study and predict the dynamic implications of managerial policies and procedures on the software development process in a variety of areas. Examples of experiments performed and insights gained include:

1. Evaluation of Schedule Estimation Alternatives.

In (Abdel-Hamid and Madnick, 1986) we investigated ways in which schedule estimation techniques are evaluated. For example, consider the case of a software project for which two software estimation techniques have been proposed. Technique A predicts 2,000 man-days, technique B predicts 3,000 man-days. The concern of top management was for accuracy of prediction. The decision was made to use technique A. When the project was completed, 2300 man-days were actually needed (15% more than predicted). In reviewing the decision to use technique A, is top management correct to conclude that it was, in fact, the more accurate estimator?

75

The answer is "not necessarily." If the project had been initiated with the 3,000 man-day estimate of technique B, there would be a difference in the hiring plans, pressures, and perceptions that affect how people behave on the project. The study results indicated that conducting the project with the 3,000 man-day estimate would have resulted in a 2,800 man-days (9% less than prediction).

The experimental results demonstrated how "a different schedule creates a different project." That is, if a software project were conducted twice using two different initial schedule estimations, the outcomes could be significantly different in nature (e.g., in terms of the actual completion time, workforce pattern, productivity). This also indicates that one can not necessarily judge a new software estimation tool strictly on the basis of how accurately it estimates historical projects.

2. Uncovering dysfunctional consequences of some currently adopted policies.

In (Abdel-Hamid and Madnick, 1986) we investigated the project scheduling practices in a major U.S. minicomputer manufacturer. In the particular organization, software project managers use Boehm's (1981) COCOMO model to come up with initial project estimates, which are then adjusted upwards using a judgemental "safety factor" to come up with the project estimates actually used. The purpose of the experiment was to investigate the implications of this safety factor policy.

Our results confirmed the value of the safety factor policy in improving estimation accuracy. One simulation showed that the estimation error which would have been 38% had been reduced to 9% by this policy. But, as noted in the first experiment, the different estimates created different projects. In this case, the safety factor policy resulted in a 43% cost increase. Thus, this more accurate estimation technique results in more costly projects. Which technique is "better"? It depends upon whether "better" means more

accurate or least costly. For the first time management had a realization of the "cost" of their more accurate schedule estimation policy.

3. Provide support for management decision making.

In (Abdel-Hamid, 1988b), we investigated the cost effectiveness of various levels of software quality assurance on total development costs. Increasing the amount of quality assurance (QA) activities directly adds new costs which are presumably more than compensated by a reduction in rework and testing. Is there an "optimal" amount of QA for a project?

Effects considered in this experiment included error generation rate factors, such as schedule pressures and phase of project, and error detection factors, such as productivity, error types, error density. The results showed that even with the same initial man-day requirements estimation, the percent of the effort allocated to QA dramatically affects the actual man-days used to complete the project. In one case studied, the resulting total project cost ranged from 3,770 man-days to 5,000 man-days with the optimal QA allocation being 16% of the development effort.

This investigation was particularly important since it provides the first capability to quantitatively analyze the costs and benefits of QA policies beyond controlled experimentation, which is usually very costly and too time consuming to be practical.

4. Providing new insights into software project phenomena.

One such phenomenon we examined in (Abdel-Hamid, 1988a) is "Brooks' Law," which states that adding manpower to a late software project makes it later (Brooks, 1978). Brooks' Law has been widely cited in the literature, but how widely applicable is it?

Our objective in this experiment was to investigate the applicability of Brooks' law to the environment of medium-sized application-type software projects (i.e., projects that are 15-100 KDSI in size). The experimental results showed that while adding more people to a late project of this type

does cause it to become more costly, it does not always cause it to complete later. The increase in the cost of the project is caused by the increased training and communication overheads, which in effect decrease the average productivity of the workforce and thus increase the project's cost in man-days. For the project's schedule to also suffer, the drop in productivity must be severe enough and late enough in the project's lifecycle to render an additional person's net_cumulative contribution to the project to be, in effect, a negative contribution. Our experimental results indicate that this happens only where management's willingness to add new staff members persists until the very final stages of the testing phase.

5. Other Experiments.

Some of the other experiments conducted investigated issues such as:

- the portability of software estimation models (e.g., why do many estimation models fail to retain their level of accuracy when transported from the originating organization to another organization?)
- the implications of staffing assignment alternatives (e.g., is it better to assign people full-time to a project or part-time?)
- the impact of different effort distributions among project phases (e.g., should the division of effort between development and testing be 80:20 or 60:40 percent?)
- the reasons for and implications of the differences between potential productivity, actual productivity, and perceived productivity.
- the basis for the "90% syndrome" phenomenon (partially explained in the Control Subsystem section of this paper).

CONCLUSION

The objective of this research effort is to enhance our understanding of

the software development process and how it is managed. To achieve this, we adopted an integrative system dynamics view of the software development process. There are three principal features that characterize our research paradigm, which we would like to reiterate in these concluding remarks. First, we emphasize the integrative perspective. We have attempted to demonstrate how the software management system is a conglomerate of interrelated and interdependent functions. Action taken by one subsystem (e.g., human resource management) can be traced throughout the entire management system (e.g., software production, planning, and control). Furthermore, the behavior of an individual subsystem in isolation may be very different from its behavior when it interacts with other subsystems (e.g., the lesson of Brooks' Law).

Second, our research approach is grounded in the feedback systems principles of system dynamics. Feedback processes are universal in social systems in general, and we attempted to show how, when applied to software project management, they do provide a powerful lens to view and understand software project behavior. It is no wonder, then, that

"most (software) managers get into trouble because they forget to think in circles. I mean this literally. Managerial problems persist because managers continue to believe that there are such things as unilateral causation, independent and dependent variables, origins, and terminations" (Weick, 1979).

Simulation, the third principal feature, is an effective tool to handle the high complexity of such integrative feedback models. Simulation's particular advantage is its greater fidelity in modeling processes, making possible both more complex models and models of more complex systems.

It also allows for vicarious experimentation. Controlled experiments in the area of software development tend to be costly and time consuming (Myers, 1978). Furthermore, even when it can be afforded "... the isolation of the effect and the evaluation of the impact of any given practice within a large,

3-7

complex and dynamic project environment can be exceedingly difficult" (Glass, 1982). In addition to permitting less costly and less time-consuming experimentation, simulation-type models make "perfectly" controlled experimentation possible.

Finally, we can indeed confirm that the very process of constructing such a simulation model can be useful in several other ways (Schultz and Sullivan, 1972):

1. Confrontation --- vague generalizations crumble when put to the test of modeling.
2. Explication --- assumptions must be made explicit, logical, and precise in order to build a simulation model.
3. Expansion --- the tendency to a holistic approach in simulation forces a broadening of one's horizon, a looking into other relevant fields for ideas. And
4. Communication --- problem-oriented simulation lead to jumping of disciplinary boundaries, less parochialism.

BIBLIOGRAPHY

1. Abdel-Hamid, T.K. "The Dynamics of Software Development Project Management: An Integrative System Dynamics Perspective." Unpublished Ph.D. dissertation, Sloan School of Management, MIT, January, 1984.
2. Abdel-Hamid, T.K. "The Dynamics of Software Project Staffing: A System Dynamics Based Simulation Approach." Accepted for publication in IEEE Transactions on Software Engineering, 1988a.
3. Abdel-Hamid, T.K. "The Economics of Software Quality Assurance: A Simulation-Based Case Study." Accepted for publication in the MIS Quarterly, 1988b.
4. Abdel-Hamid, T.K. and Madnick, S.E. "Impact of Schedule Estimation on Software Project Behavior." IEEE Software, (July, 1986).
5. Abdel-Hamid, T.K. and Madnick, S.E. Software Project Management. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., to be published in 1988.
6. Baber, R.L. Software Reflected. New York: North Holland Publishing Company, 1982.

7. Basili, V.R. "Improving Methodology and Productivity through Practical Measurement." A lecture at the Wang Institute of Graduate Studies, Lowell, Mass., Nov., 1982.
8. Beck, L.L. and Perkins, T.E. "A Survey of Software Engineering Practice: Tools, Methods, and Results." IEEE Transactions on Software Engineering, Vol. SE-9, No. 5, (Sept., 1983).
9. Boehm, B.W. Software Engineering Economics, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1981.
10. Brooks, F.P. The Mythical Man Month. Reading, Mass: Addison-Wesley Publishing Co., 1978.
11. Buckley, F. and Poston, R. "Software Quality Assurance." IEEE Trans. on Software Engineering, (January, 1984), 36-41.
12. Chrysler, E. "Some Basic Determinants of Computer Programmer Productivity." Communications of the ACM, Vol. 21, No. 6, (June, 1978), 472-483.
13. Cooper, J.D. "Corporate Level Software Management." IEEE Transactions on Software Engineering, Vol. SE-4, No. 4, July, 1978.
14. Cougar, J.D. and Zawacki, R.A. Motivating and Managing Computer Personnel. New York: John Wiley & Sons, Inc., 1980.
15. DeMarco, T. Controlling Software Projects. New York: Yourdon Press, Inc., 1982.
16. DePree, R.W. "The Long and Short of Schedules." Datamation, (June 15, 1984), 131-134.
17. Forrester, J.W. Industrial Dynamics. Cambridge, Mass: The MIT Press, 1961.
18. Freedland, M. "What You Should Know About Programmers." Datamation, March 15, 1987, pp. 91-102.
19. Ibrahim, R.L. "Software Development Information System." Journal of Systems Management, (Dec., 1978), 34-39.
20. Jensen, R.W. and Tonies, C.C. Software Engineering. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1979.
21. Kelly, T.J. "The Dynamics of R&D Project Management." Unpublished M.S. Thesis, MIT, Sloan School of Management, Cambridge, Massachusetts, 1970.
22. McKeen, J.D. "Successful Development Strategies for Business Application Systems." MIS Quarterly, Sept., 1983.
23. Mills, H.D. Software Productivity. Canada: Little, Brown & Co., 1983.
24. Musa, J.D. "Software Engineering: The Future of A Profession." IEEE Software, January, 1985, 55-62.

25. NASA. "Software Development History for Dynamics Explorer (DE) Attitude Ground Support System (AGSS)." GSFC Code 580, June, 1983.
26. Nay, J.N. "Choice and Allocation in Multiple Markets: A Research and Development Systems Analysis." Unpublished M.S. Thesis, MIT, Dept. of Electrical Engineering, Cambridge, Massachusetts, 1965.
27. Newport, John P. Jr. "A Growing Gap in Software." Fortune, April 28, 1986, 132-142.
28. Pooch, U.W. and Gehring, P.F. Jr. "Toward a Management Philosophy for Software Development." In Advances in Computer Programming Management. Edited by T.A. Rullo. Philadelphia, PA: Heyden & Sons, Inc., 1980.
29. Radice, A. "Productivity Measures in Software." The Economics of Information Processing: Operations, Programming, and Software Models. Volume II. Edited by R. Goldberg and H. Lorin. New York: John Wiley & Sons, Inc., 1982.
30. Ramamoorthy, C.V. et al. "Software Engineering: Problems and Perspectives." Computer, Oct., 1984.
31. Richardson, G.P. and Pugh, G.L. III. Introduction to System Dynamics Modeling with Dynamo. Cambridge, Mass: The MIT Press, 1981.
32. Roberts, E.B. The Dynamics of Research and Development. New York: Harper & Row, 1964.
33. Roberts, E.B. (ed.). Managerial Applications of System Dynamics. Cambridge, Mass: The MIT Press, 1981.
34. Schein, E.H. Organizational Psychology. 3rd edition. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1980.
35. Scott, R.F. and Simmons, B.D. "Programmer Productivity and the Delphi Technique." Datamation, (May, 1974), 71-73.
36. Shell, R.L. "Work Measurement for Computer Programming Operations." Industrial Engineering, (Oct., 1972), 32-36.
37. Shooman, M.L. Software Engineering - Design, Reliability and Management. New York: McGraw-Hill, Inc., 1983.
38. Steiner, I.D. Group Process and Productivity. New York: Academic Press, 1972.
39. Tausworthe, R.C. Standardized Development of Computer Software. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1977.
40. Thayer, R.H. "Modeling a Software Engineering Project Management System." Unpublished Ph.D. dissertation, University of California, Santa Barbara, 1979.
41. Thayer, R.H. et al. "Major Issues in Software Engineering Project Management." IEEE Transactions on Software Engineering, Vol. SE-7, No. 4, July, 1981.

42. Thomsett, R. People Project Management. New York: Yourdon Press, Inc., 1980.
43. Weick, K.E. The Social Psychology of Organization. 2nd edition. Reading, Mass: Addison-Wesley Publishing Co., Inc., 1979.
44. Weinberg, G.M. Understanding the Professional Programmer. Boston: Little, Brown & Co., Inc., 1982.
45. Weiss, D.M. "Evaluating Software Development by Error Analysis." Journal of Systems and Software, Vol. 1, 1979, 57-70.
46. Zelkowitz, M.V. "Perspectives on Software Engineering." Computing Surveys, Vol. 10, No. 2, (June, 1978).
47. Zmud, R.W. "Management of Large Software Development Efforts." MIS Quarterly, Vol. 4, No. 2, June 1980, 45-56.

Acknowledgements

We appreciate the contribution of each of the individuals in the organizations providing perspectives and data to this research effort. In addition, we thank Robert Zmud, Chris Kemerer, and the anonymous reviewers, whose suggestions have improved this article's organization and readability. Work reported herein was supported, in part, by NASA research grant NAGW-448.

Date Due

	MAR. 28 1994
	SEP. 06 1994
SEP 15 1996	
MAY 12 '89	MAY 31 1997
	APR 06 1998
DEC 27 1989	
MAR 17 1990	
MAY 12 1990	
AUG 02 1991	
MAR 2 1991	
NOV 22 1991	

MIT LIBRARIES DUPL
3 9080 005 233 215

